THE UNITED STATES PATENT AND TRADEMARK OFFICE 90/006403

U.S. Patent No.: 5,930,798

Issued: July 27, 1999

Title: UNIVERSAL DATA MEASUREMENT, ANALYSIS AND CONTROL SYSTEM

Reference No.: 1005 – 0001

March 17, 2003

BOX REEXAM
COMMISSIONER FOR PATENTS
Washington, DC 20231

REPLY TO PATENT OWNER'S STATEMENT SUBMITTED IN THE REEXAMINATION OF U.S. PATENT NO. 5,930,798

Dear Sir:

This Reply is submitted in accordance with 37 C.F.R. § 1.535 within two months of the Patent Owner's Statement of January 17, 2003. Furthermore, this Reply has been served upon the patent owner in accordance with 37 C.F.R. §1.248. Please change the correspondence address of the third part Requestor to:

TOLER, LARSON & ABEL, L.L.P.
ATTN: JEFF TOLER
5000 PLAZA ON THE LAKE, SUITE 265
AUSTIN, TEXAS 78755-9567

I. Summary

The Requestor reiterates arguments made in the granted Request for Reexamination, dated October 7, 2002, of U.S. Patent No. 5,930,798 (hereafter the '798 patent). The prior art cited in the Request for Reexamination is a single reference that teaches each of the claimed elements arranged as in the claim, expressly or inherently, as interpreted by one of ordinary skill

in the art. In addition, the '798 Patent would be rendered obvious as the one or more references that were available to the inventor either teach a suggestion to combine or modify the reference(s) such that the combination or modification would appear to be sufficient to have made the claimed invention obvious to one of ordinary skill in the art. Nothing in the Patent Owner's Statement has overcome the Requestor's position presented in the Request for Reexamination filed October 7, 2002.

II. Reply to Issues Raised in Brief Summary of Patent Owner's Statement

Requestor disagrees with the remarks of the Owner regarding the merit and nature of the arguments made in the Request for Reexamination. The arguments have merit, are not "ludicrous", and do raise an issue of patentability as evidenced by the USPTO's action of granting of the *ex parte* Reexamination.

With regard to the submitted publication, "Building Effective Decision Support Systems" by Ralph H. Sprague, Jr. and Eric D. Carlson ("the DSS Reference"), the DSS Reference is an enabling anticipatory prior art reference with respect to the '798 patent. A reference or publication is prior art for all that it teaches (Beckman Instruments v. LKB Produkten AB, 892 F. 2d 1547, 1551, 12 USPD 2d 1301, 1304 (Fed. Cir. 1989)). "All of the disclosures in a references must be evaluated for what they fairly teach one of ordinary skill in the art." (In re Lemelson, 397 F.2d 1006, 1009, 158 USPQ 275, 277 (CCPA 1968)). Moreover, "[t]o be enabling prior art, [a] printed publication need only enable ... only one species covered by [a] broad patent claim." (Johns Hopkins University v. CELLPRO, 894 F. Supp. 819 (D. Del. Aug 04, 1995) affirmed by Johns Hopkins University v. CELLPRO, 978 F. Supp. 184). The Owner's argument that elements of the claims are found in various locations within the DSS Reference does not invalidate its use in a 35 U.S.C. §102 or §103 rejection. One should expect that in a teaching reference such as the DSS Reference, definitions and explanations of concepts are often presented before the disclosure of more complex or aggregate concepts. In fact, the charts presented by the Owner make this point. Many of the references occur in a smaller set of later chapters, periodically referencing definitions and explanations provided in earlier chapters. Therefore, the Owner's argument regarding citation frequency does not overcome the *prima facie* case of anticipation as established in the Request for Reexamination.

In addition, the Owner states that the DSS Reference "cannot be said to teach" the invention claimed in the '798 patent. The Owner justifies this statement by stating that the re-examination references to the "database management subject areas" are not applicable since the subject of the patent "is not database management". The topic of the DSS Reference is clearly not solely limited to database management. The section on database management describes commonly practiced methods for representing essential elements of a DSS. As such, the Owner's statement regarding the scope of the DSS Reference relative to the scope of the '798 patent does nothing more than attempt to mischaracterize the text without any supporting rational. In addition, the claims of the '798 patent do not specifically exclude databases or the subject of database management. The DSS Reference and other similar references are therefore relevant to this Reexamination.

III. Reply to Arguments Regarding Claims 1-20 in the Patent Owner's Statement

On Page 6 of the Owner's Statement, the Owner noted that an "example of the claimed hierarchical index is disclosed in the specification as a 'Measurement Object Model,' which 'organized objects of a CASE [Computer Aided Software Engineering] tool file or database and associates a set of measurements to each of these objects." While CASE may be an example, the Summary (Column 3, Line 36), the Detailed Description (Column 4, Line 56), and the preamble to independent Claim 1 cite a universal data analysis, measurement and control system. The DSS Reference recites many examples that are covered under the scope of universal data analysis, measurement and control systems.

Specifically, the DSS Reference teaches a system that can be applied universally to a wide range of applications, as in: 1) the development, maintenance, and reuse of software (p. 280); 2) financial and capital investment analysis (p. 5); 3) airline operation schedules, capacities and forecasts (p. 5); 4) manufacturing production scheduling within a paper company; 5) executive information system (p. 5); 6) analysis of police services and scheduling zones by the city of San Jose (p. 41); and 7) a matrix of DSS applications from structured to unstructured and multi-organizational uses, depicted in Fig. 4-1, (p. 96).

Addressing the area of analysis, the DSS Reference teaches that a DSS can be used for cognitive testing (p. 163), systems measurement (p. 164), systems analysis (p. 164), cost/benefit analysis (p. 165), value analysis (p. 166), multiple analysis techniques (p. 167), and a general model for analysis through measurement and data evaluation (p. 167). Addressing the area of measurement, the DSS Reference teaches taxonomy of what to measure (p. 158), techniques for how to measure (p. 159), and provides examples of measurements (p. 160 Fig. 6-6). Addressing a variety of types of input data, the DSS Reference describes a variety of types of data as shown by the intermixing of software components (p. 281 and Fig. 10). Finally, the DSS Reference teaches that the data used in decision-making is likely to come from both internal and external sources (p. 241).

Furthermore, in the definition of "Measurement Object Model" in the '798 patent, the phrase "organizes the measurable objects" is associated with " a CASE" or "database." Therefore, given the scope of the claims of the '798 patent and the definitions provided by the Owner, the DSS Reference is analogous art. As such, the term "instantiate" is reasonably defined in relation to a database or a programming tool. The Owner's statement provides "instantiation" is 'Producing a more defined version of some object by replacing variables with values." In the context of a database, a definition of "instantiate" is to create an entry in a database table. ("Instantiation", Whatis.com, 3/7/2003). In programming, a sample definition of "instantiate" is to create a concrete instance of a class, create an object from a class, or declare an object of a given data type such as a class. ("Teach Yourself C++, Fourth Edition" by Al Stevens, MIS Press, New York, 1995.; "Instantiation", Whatis.com, 3/7/2003).

Instantiating an index such as a relational database was known in the art more than one year prior to the filing date of the '798 patent, as shown in the DSS Reference (Chapter 8). For a set of related data models to be useful, the data models would include the ability to receive and store data (i.e. to be instantiated). Further, instantiating an index, such as an object or model, was known in the art more than one year prior to the filing date of the '798 patent, as shown in the DSS Reference (Chapter 9) and other references ("Teach Yourself C++, Fourth Edition" by Al Stevens, MIS Press, New York, 1995).

With regards to the Owner's Statement regarding the step of comparing in claim 1, the DSS Reference clearly describes "comparing at least one of the data sets linked to at least one of said instantiated indexes to at least another of the data sets linked to at least another of said instantiated indexes by means of the programmed computer." In addition to the arguments presented in the Request for Reexamination, comparing data is well known in both database art and data modeling art. Such a comparing step is used for sorting and querying functionality common to databases at the time of publication of the DSS Reference. To query and sort, data sets in a table (i.e. instantiated model) must be compared. As such, the claim element of "comparing at least one of the data sets linked to at least one of said instantiated indexes to at least another of the data sets linked to at least another of said instantiated indexes by means of the programmed computer" is both expressly and inherently described in the DSS Reference. Such a comparing step is also well known in the programming arts. Subsets of instantiated objects are often compared in the operation of a program such as an IF statement calling two instantiated objects for comparison. (pp 168-169. "Teach Yourself C++, Fourth Edition" by Al Stevens, MIS Press, New York, 1995).

In summary, instantiation of a hierarchical index as claimed in the '798 patent was well known in the art. Citations regarding databases and model programming describe the step of instantiation, comparison of instantiations, and storage of instantiated models.

The above Reply addresses the Owner's arguments that appear to only relate to Claim 15. No arguments in the Owner's Statement appear to address other claims. As such, the Requestor submits that all of the claims of the '798 patent lack novelty and are obvious based on the cited prior art as explained in the Request for Reexamination filed October 7, 2002.

VI. Conclusion

For at least the foregoing reasons, it is submitted that claims 1-20 of the '798 Patent are not novel and are obvious in light of the cited prior art. Continued Reexamination of the '798 Patent and a determination of unpatentability is therefore respectfully requested.

The Director is hereby authorized to credit any overpayments or charge any additional fees regarding this Reply to Patent Owner's Statement to Deposit Account No. 50-2469.

CERTIFICATE OF MAILING UNDER 37 CFR § 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service via Express Mail , in an envelope addressed to Commissioner for Patents, Washington, D.C. 20231 on the date shown below.

Katrina Prati — 3/17/03
Katrina Prati — Date

EXPRESS MAIL LABEL NO:
ET 092 082 768 US

Respectfully submitted,

Jeffrey G. Toler
Reg. No. 38,342

Customer Number 34456
TOLER, LARSON & ABEL, L.L.P.
P.O. BOX 29567
AUSTIN, TEXAS 78755-9567

Telephone: 512-327-5515
Facsimile: 512-327-5452

34456
PATENT TRADEMARK OFFICE

EXHIBIT A

"instantiation," Whatis.com (www.whatis.com) , July 30, 2002.

instantiation

In programming, instantiation is the creation of a real *instance* or particular realization of an abstraction or template such as a class of objects or a computer process. To instantiate is to create such an instance by, for example, defining one particular variation of object within a class, giving it a name, and locating it in some physical place.

1) In object-oriented programming, some writers say that you instantiate a class to create an object, a concrete instance of the class. The object is an executable file that you can run in a computer.

2) In the object-oriented programming language, Java, the object that you instantiate from a class is, confusingly enough, called a class instead of an object. In other words, using Java, you instantiate a class to create a specific class that is also an executable file you can run in a computer.

3) In approaches to data modeling and programming prior to object-oriented programming, one usage of *instantiate* was to make a real (data-filled) object from an abstract object as you would do by creating an entry in a database table (which, when empty, can be thought of as a kind of class template for the objects to be filled in).

Read more about it at:
> SearchVB.com has resources for this and other Visual Basic programming terms.

This word suggested by: John MacLeod
Last updated on: Jul 30, 2002

Learn more about:
> XML Learning Guide

- Site Map for whatis.com
- Quiz #36: Linux basics
- Got a favorite blog? Send us the URL!
- IT Crossword Puzzle
- Discussion Forums
- Recently added/updated
- Order the whatis.com book

Email a friend:

Send this page to a friend! Email a friend

Note: Email addresses will only be used to send site content to your friend(s).

Whatis.com is part of the TechTarget network of industry-specific IT Web sites

APPLICATIONS
SearchCRM.com
SearchSAP.com

DEVELOPMENT
SearchVB.com

ENTERPRISE IT MANAGEMENT
SearchCIO.com

CORE TECHNOLOGIES
SearchDatabase.com
SearchNetworking.com
SearchSecurity.com
SearchStorage.com
SearchSystemsManagement.com
SearchWebServices.com
Whatis.com

PLATFORMS
Search390.com
Search400.com
SearchDomino.com
SearchEnterpriseLinux.com
SearchSolaris.com
SearchWin2000.com
SearchWindowsManageability.com

TechTarget
The Most Targeted IT Media

TechTarget Enterprise IT Conferences | TechTarget Corporate Web Site | Media Kit

Explore **SearchTechTarget.com**, the guide to the TechTarget network of industry-specific IT Web sites.

EXHIBIT B

Stevens, Al, "teach yourself C++, Fourth Edition" MIS Press, New York, 1995.

teach yourself...

FOURTH EDITION

Al Stevens

A Subsidiary of
Henry Holt and Co., Inc.

First Edition—1995

Library of Congress Cataloging-in-Publication Data

Stevens, Al,
Teach yourself--C++ / Al Stevens. --4th ed.
p. cm.
Includes index.
ISBN 1-55828-406-0
1. C++ (Computer program language) I. Title.
QA76.73.C153S73 1994
005.13'3--dc20 94-24954
CIP

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

MIS:Press books are available at special discounts for bulk purchases for sales promotions, premiums, fund-raising, or educational use. Special editions or book excerpts can also be created to specification.

For details contact: Special Sales Director
MIS:Press
a subsidiary of Henry Holt and Company, Inc.
115 West 18th Street
New York, New York 10011

Editor-in-Chief: Paul Farrell

Managibg Editor: Cary Sullivan

Development Editor: Debra Williams Cauley

Technical/Copy Editor: Betsy Hardinger

Production Editor: Anthony Washington

load operators, always try to overload them so that they perform operations that resemble their use with intrinsic data types in the C++ language.

RELATIONAL OPERATORS

Suppose you want to compare dates. Perhaps you need to use an expression such as the following one:

```
if (newdate < olddate)
// ....
```

You can overload relational operators in the same way that you overloaded the addition operator.

Exercise 8.3 shows the *Date* class with overloaded operators that compare dates.

Exercise 8.3 *Overloading Relational Operators*

```
#include <iostream.h>
class Date {
    int mo, da, yr;
public:
    Date(int m=0, int d=0, int y=0)
        { mo = m; da = d; yr = y; }
    void display()
        { cout << mo << '/' << da << '/' << yr; }
    // ----- overloaded operators
    int operator==(Date& dt);
    int operator<(Date&);
};
// ----- overloaded equality operator
int Date::operator==(Date& dt)
{
    return (this->mo == dt.mo &&
            this->da == dt.da &&
            this->yr == dt.yr);
}
```

continued

Exercise 8.3 *Overloading Relational Operators (continued)*

```
// ----- overloaded less than operator
int Date::operator<(Date& dt)
{
    if (this->yr == dt.yr)    {
        if (this->mo == dt.mo)
            return this->da < dt.da;
        return this->mo < dt.mo;
    }
    return this->yr < dt.yr;
}
main()
{
    Date date1(12,7,41),
         date2(2,22,90),
         date3(12,7,41);

    if (date1 < date2)    {
        date1.display();
        cout << " is less than ";
        date2.display();
    }
    cout << '\n';
    if (date1 == date3)    {
        date1.display();
        cout << " is equal to ";
        date3.display();
    }
}
```

The *Date* class in Exercise 8.3 has two overloaded relational operators: the equal to (==) and the less than (<) operators. The *main* function declares three dates, compares them, and displays the following messages:

free store

The C++ heap. A dynamic memory pool that programs use to allocate and release temporary memory buffers.

friend

A function that has access to the private members of a class but that is not a member function of that class. The class definition declares the function to be a *friend.*

hierarchy

See "class hierarchy."

inheritance

The ability for one class to inherit the characteristics of another. The inherited class is said to be derived from the base class. Also called "subclassing".

implementation

The private members of a class. The implementation defines the details of how the class implements the behavior of the abstract base type. See also "interface."

inline function

A function that the compiler compiles as in-line code every time the function is called.

insertion operator

The overloaded << operator that writes (inserts) values to an output stream. See also "extraction operator."

instantiate

Declare an object of a data type, usually a class.